IBM Systems & Technology Group

IBM HPC Development
MPI update

Chulho Kim

Scicomp 2007

Enhancements in PE 4.3.0 & 4.3.1

- MPI 1-sided improvements (October 2006)
- Selective collective enhancements (October 2006)
- AIX IB US enablement (July 2007)

MPI 1-sided improvements

MPI 1-sided testcases provided by William Gropp and Rajeev Thakur from Argonne National Lab. These two test programs perform nearest-neighbor ghost-area data exchange:

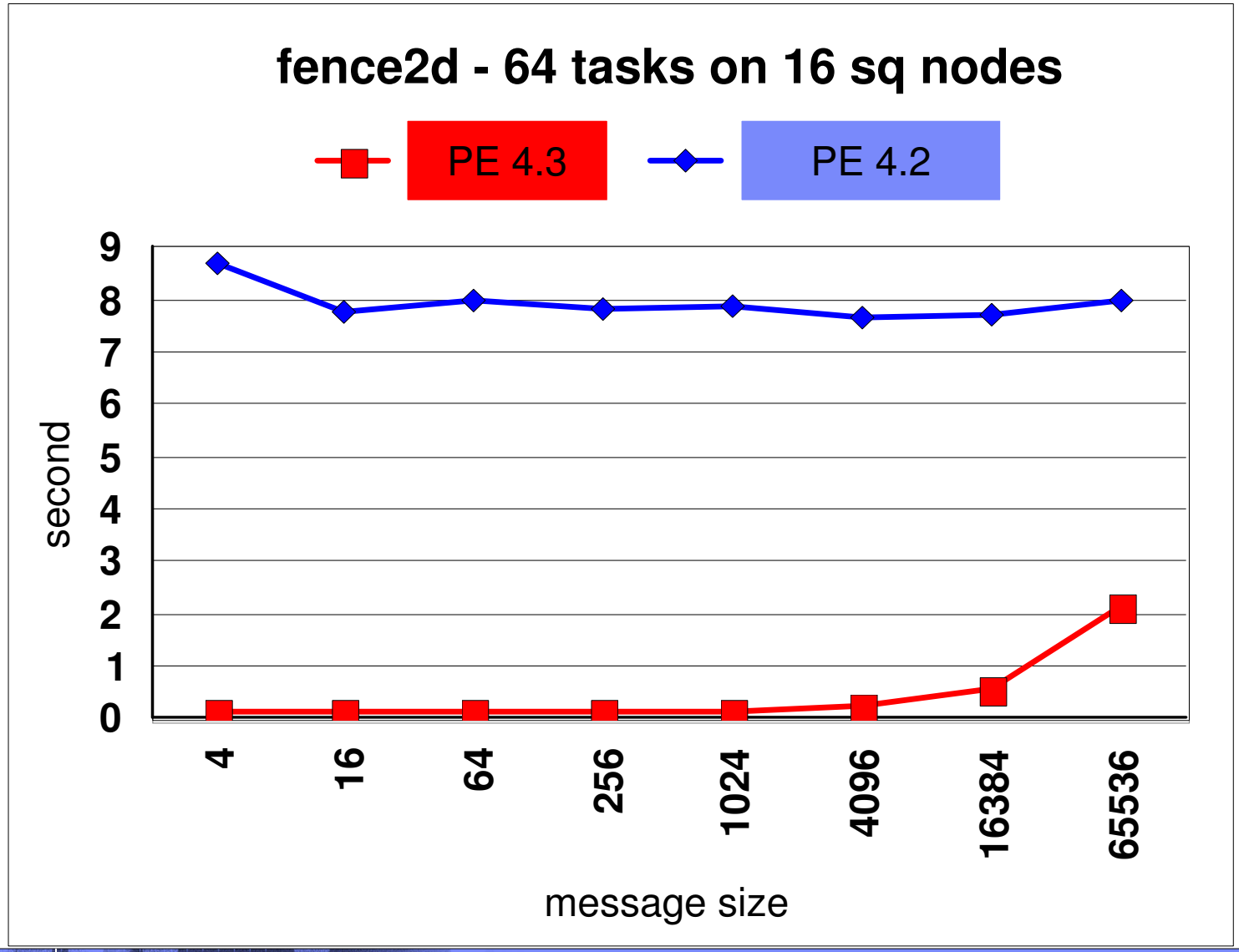
fence2d.c - four MPI_Puts with MPI_Win_fence call

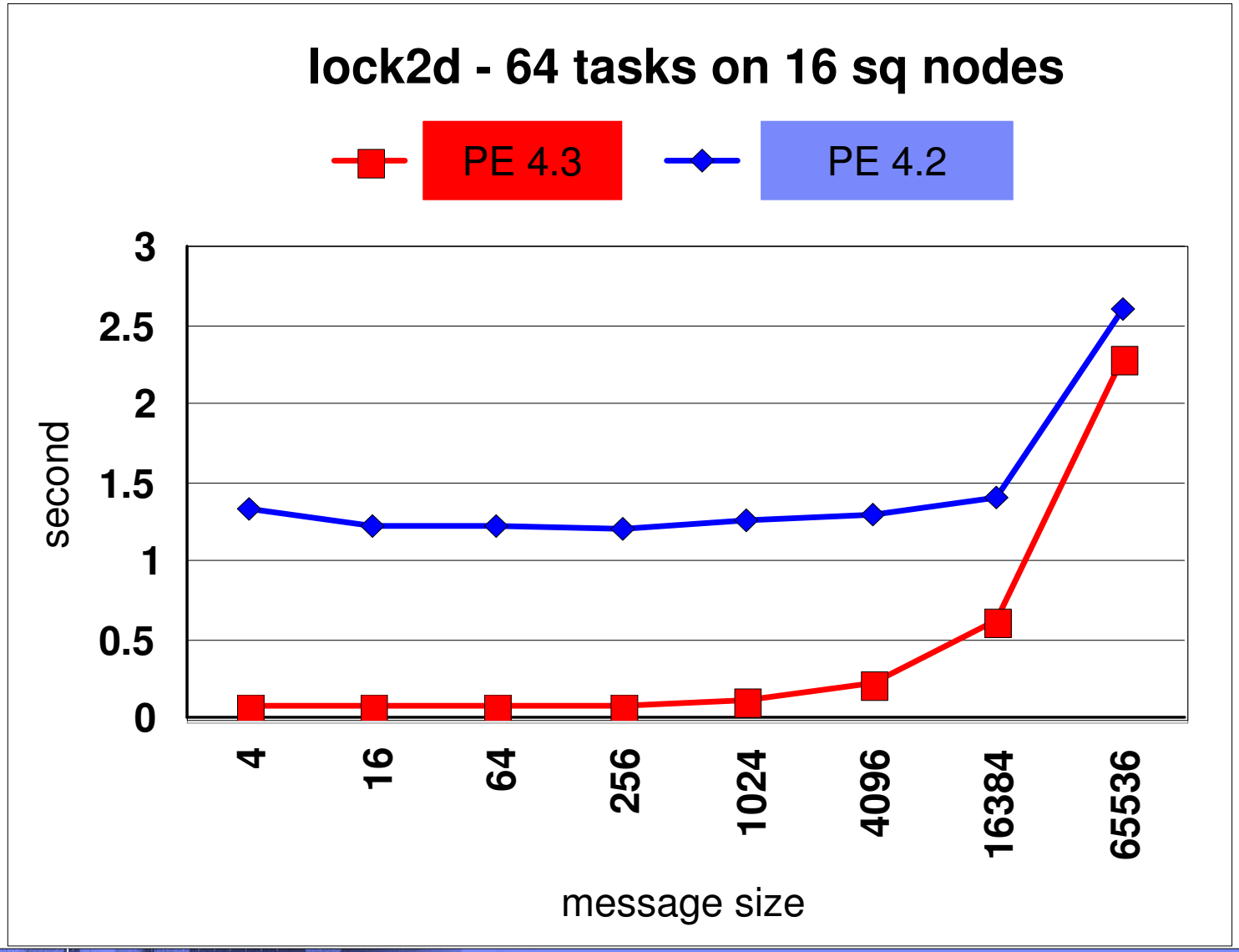
lock2d.c - four MPI_Puts with MPI_Win_lock and MPI_Win_unlock calls

A single communication step for this set includes a synchronization call to start an epoch, a MPI_Put to put the data to each of its four neighbors (4 MPI_Puts) and a synchronization call to end the epoch.

Configurations: 64 tasks – 16 SQ IH nodes

Improvements seen: > 10% and sometimes up to 90% reduction in time taken to do the operation especially in small messages (< 64K).

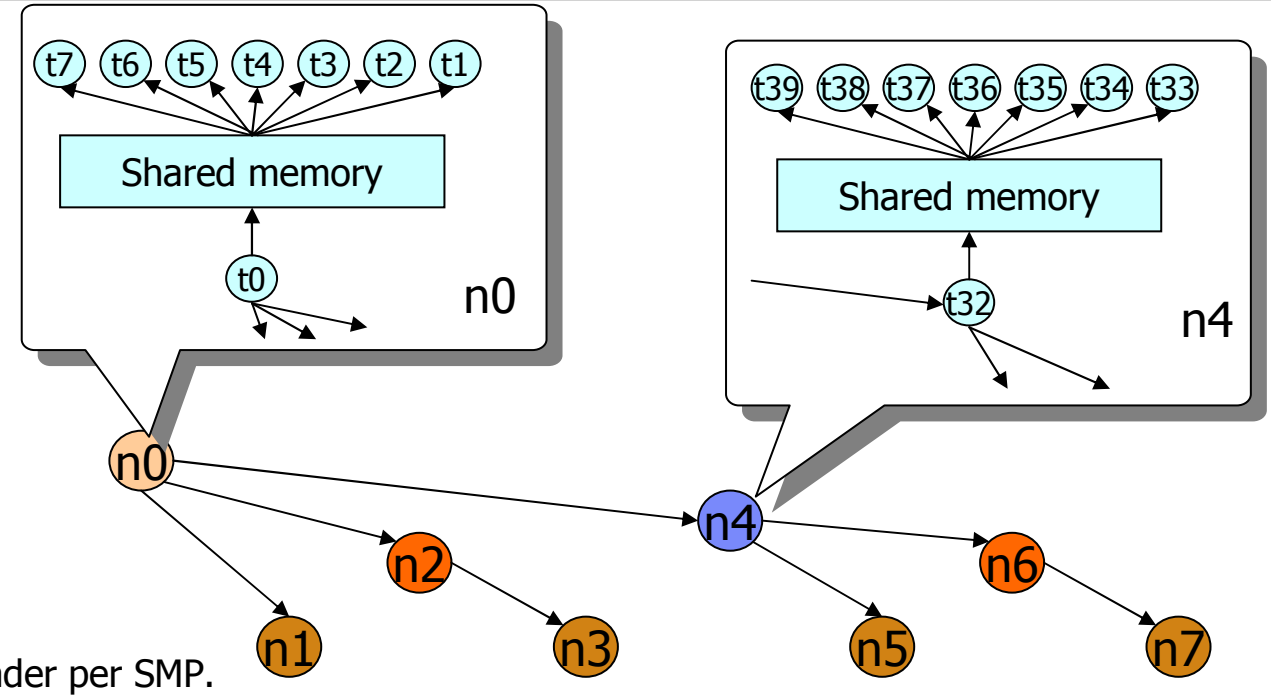




New algorithms in PE 4.3.0

- A set of high radix algorithms for:
  - MPI_Barrier,
  - small message (<= 2KB) MPI_Reduce,
  - small message (<= 2KB) MPI_Allreduce, and
  - small message (<= 2KB) MPI_Bcast.
- New parallel pipelining algorithm for large message (>= 256KB) MPI_Bcast.
- These new algorithms are implemented in IBM Parallel Environment 4.3.
- Performance comparison are made between IBM PE4.3 and IBM PE4.2.2 (old algorithms)

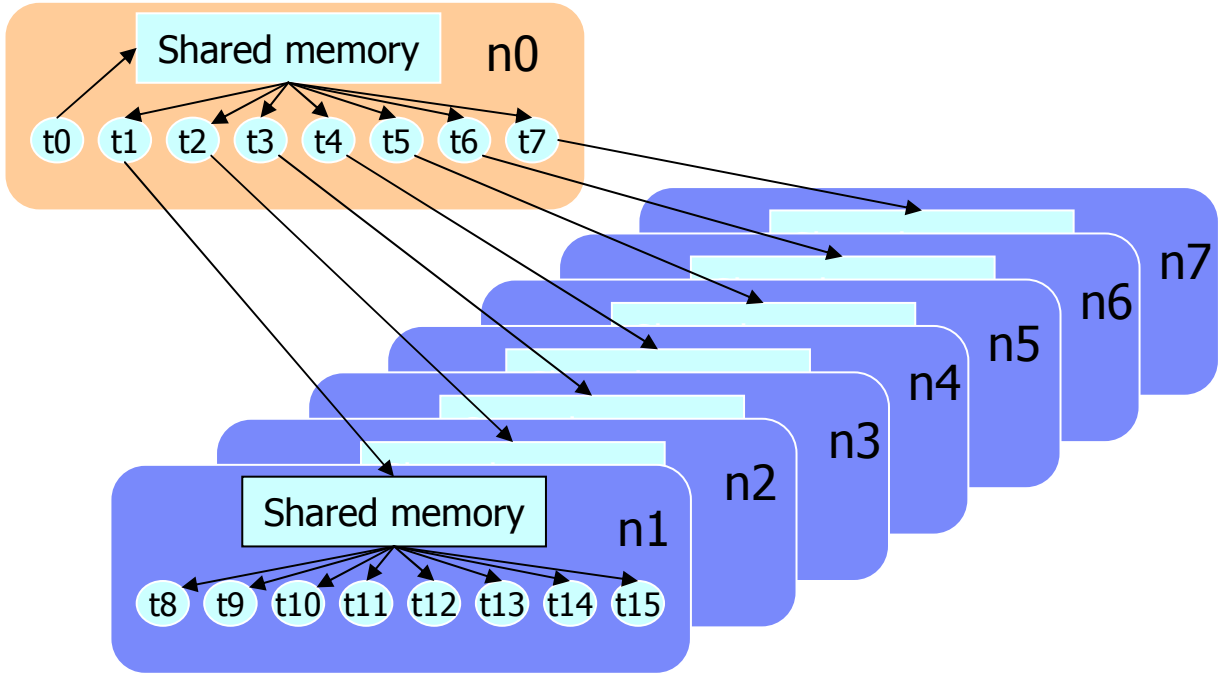
Common shared memory optimization for SMP cluster



- One task is selected as node leader per SMP.
- Possible prolog step - node leader gathers inputs from tasks on the same SMP node and forms partial result through shared memory.
- Inter-node communication steps among node leaders through interconnect.
- Possible epilog step – node leader distributes results among the tasks on the same node through shared memory.
- Benefit from shared memory is only for the intra-node part. Available communication resource may not be fully utilized.

Example of MPI_Bcast with root=t0, using the common shared memory optimization

- 8 SMP nodes each running 8 MPI tasks.
- Colors of the nodes represent the inter-node step, after which the node has the broadcast message: Root, Step 0, Step 1, Step2. Arrows show message flow.
- 3 inter-node steps are needed to complete the broadcast.

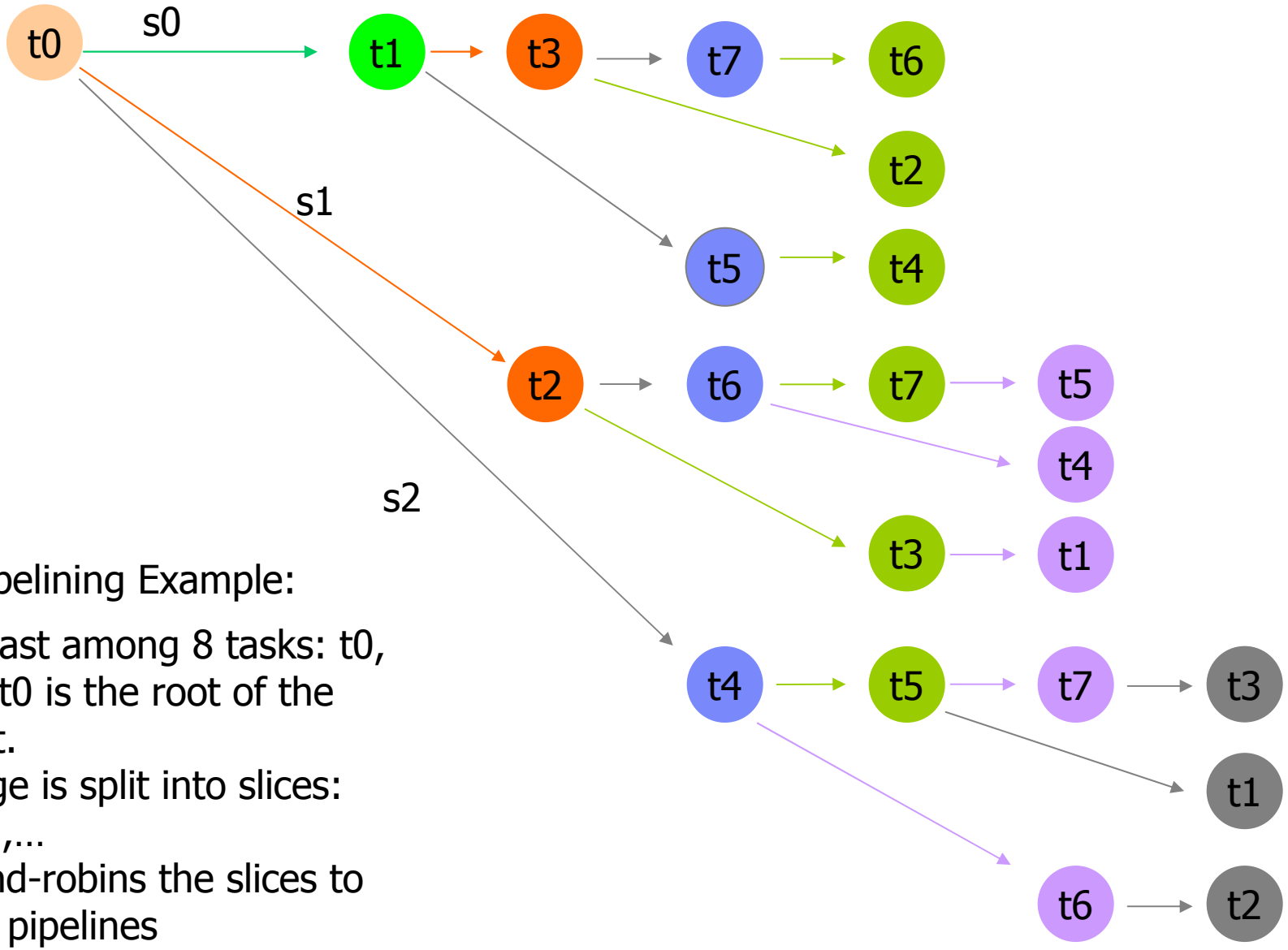


High Radix Algorithm

- k (k >= 1) tasks on each SMP participating in the inter-node communication.
- Possible shared memory step after each inter-node step to synchronize and reshuffle data.
- Better short message performance as long as the extra shared memory overhead and switch/adapter contention are low.
- 64bit only, requires shared memory and same number of tasks on each nodes.
- In this MPI_Bcast example, k = 7
- only one inter-node step is needed.

Parallel pipeline algorithm: MPI_Bcast

- Split large data into small slices and pipeline the slices along multiple broadcast trees in parallel and in non-conflicting fashion.
- Optimal on any number of tasks (power of two or not)
- Communication scheduling is through simple bit operations on rank of task. Schedule does not need to be cached in the communicator.
- Once the pipeline is established, each task is doing concurrent send and receive of data slices
- Paper accepted by EuroPVMMPI07



Parallel Pipelining Example:

- Broadcast among 8 tasks: t0, t1, … t7. t0 is the root of the broadcast.
- Message is split into slices: s0, s1, s2,…
- t0 round-robins the slices to the three pipelines

Benchmark config

- Micro benchmarks
  - on up to 32 P5 nodes and 8 tasks per node
  - AIX5.3, IBM HPS (4 links), IBM RSCT/LAPI 2.4.3, and IBM LoadLeveler 3.4
- For each benchmark,
  - run using the old algorithm in PE4.2.2 first,
  - then run with the new algorithm in PE4.3.

Benchmark

- Loops of:

```
MPI_Barrier();
Start_time = MPI_Wtime();
MPI_Barrier(); /* or other measured
      collectives */
End_time = MPI_Wtime();
MPI_Barrier();
```

Timing

- Global switch clock is used. MP_CLOCK_SOURCE=Switch
- Time of the measured collectives at any task for a particular loop is:
  - for MPI_Barrier and MPI_Allreduce
  - End_time of this task – Start_time of this task
  - for MPI_Bcast
  - End_time of this task – Start_time at the root task
  - for MPI_Reduce
  - End_time of the root task – Start_time of this task
- 1. For every loop, select the longest time reported by any task as the time of the loop 2. Select the time of the fastest loop
  - This filters out OS jitter impact and highlights the algorithm capability